Truly Distribution-Independent Algorithms for the *N*-body Problem*

Srinivas Aluru
Ames Laboratory
Iowa State University
Ames, IA 50011

G.M. Prabhu
Computer Science Dept.
Iowa State University
Ames, IA 50011

John Gustafson
Ames Laboratory
Iowa State University
Ames, IA 50011

Abstract

The N-body problem is to simulate the motion of N particles under the influence of mutual force fields based on an inverse square law. Greengard's algorithm claims to compute the cumulative force on each particle in $O(N)$ time for a fixed precision irrespective of the distribution of the particles. In this paper, we show that Greengard's algorithm is distribution dependent and has a lower bound of $\Omega(N \log^2 N)$ in two dimensions and $\Omega(N \log^4 N)$ in three dimensions. We analyze the Greengard and Barnes-Hut algorithms and show that they are unbounded for arbitrary distributions. We also present a truly distribution independent algorithm for solving the N-body problem in $O(N \log N)$ time in two dimensions and in $O(N \log^2 N)$ time in three dimensions.

1 Introduction

A large number of physical systems can be studied by simulating the interactions between the particles constituting the system. In a typical system each particle influences every other particle, often based on an inverse square law such as Newton's law of gravitation or Coulomb's law of electrostatic interaction. Examples of such physical systems can be found in astrophysics, plasma physics, molecular dynamics and uid dynamics. Since the simulation involves following the trajectories of motion of a collection of N particles, the problem is termed the N-body problem. Apart from traditional applications in the study of physical systems, some problems in numerical complex analysis and elliptic partial differential equations can also be solved using this approach. Applications of the problem are also found in the radiosity method, which attempts to create images by computing the equilibrium distribution of light for complex scene geometries.

Since it is not possible to solve the equations of motion for a collection of four or more particles in closed form, iterative methods are used to solve the N-body problem. At each discrete time interval, the force on each particle is computed and this information is used to update the position and velocity of each particle. A straightforward computation of the forces requires $O(N^2)$ work per iteration. The rapid growth with N effectively limits the number of particles that can be simulated by this method.

Several approaches have been used to reduce the complexity per iteration. Some of the techniques include transforming the problem to a position-velocity phase space, imposing a grid on the system of particles and computing cell-cell interactions. Such techniques either fail to model the system accurately or degrade to $O(N^2)$ complexity for non-uniform distributions of the particles.

Recently, a new class of particle simulation methods have emerged to solve the N-body problem. These methods are characterized by an organization of the particles into a hierarchy of clusters, starting from a cluster containing all the particles to clusters containing the individual particles. These methods are usually referred to as *hierarchical methods* or *tree methods*. Such a method was first proposed by Appel [2], whose scheme allows for clusters with arbitrary shapes.

A number of algorithms followed the work of Appel. Widely respected among these are the Barnes- Hut [4] and the Greengard [7] methods. Both depend on a data structure constructed by a fixed hierarchical cubical subdivision of the space. Salmon [14] studies the Barnes-Hut algorithm in great detail. While Appel and Barnes-Hut achieve required accuracy by restricting which clusters may interact, Greengard uses *multipole expansions* [7] to approximate the interactions to the desired precision. With the notable exception of Greengard, most researchers paid little attention to a rigorous worst-case complexity analysis of their algorithms. Greengard claims his algorithm reduces the complexity to $O(N)$

*This work is supported by the Applied Mathematical Sciences Program of the Ames Laboratory-USDOEunder contract No. W-7405-ENG-82.

per iteration.

In this paper, we show that the Greengard's algorithm is not $O(N)$, as claimed. Both Barnes-Hut and Greengard's methods depend on the same data structure, which we show is distribution-dependent. For the distribution that results in the smallest run- ning time, we show that Greengard's algorithm is $O(N \log^2 N)$ in two dimensions and $O(N \log^4 N)$ in three dimensions. Both algorithms are unbounded for arbitrary distributions.

We have designed a hierarchical data structure whose size depends entirely upon the number of particles and is independent of the distribution of the particles. Both Greengard's and Barnes-Hut algorithms can be used in conjunction with this data structure to reduce their complexity. Apart from reducing the complexity of the Barnes-Hut algorithm, the data structure also permits more accurate error estimation. The multipole algorithm designed using this data structure has a complexity of $O(N \log N)$ in two dimensions and $O(N \log^2 N)$ in three dimen sions. To the best of our knowledge, this is the fastest distribution-independent algorithm for the N-body problem.

The rest of the paper is organized as follows: In Section 2, we analyze the complexity of Greengard and Barnes-Hut algorithms. We derive lower and upper bounds on the data structure used in these algorithms and use this result to disprove claims on their complexity. Section 3 contains the description of our new hierarchical data structure. We also show how to use the Greengard and Barnes-Hut algorithms on this data structure. An algorithm to create this new data structure is described in Section 4.

2 The complexity of Greengard and Barnes-Hut Algorithms

The Greengard and Barnes-Hut methods for computing N-body interactions consist of two alternating phases, repeated every time step:

1. Computing a hierarchical tree data structure with the leaves representing the particles and the root of the tree representing the entire system.

2. Traversing this data structure to compute the force on each particle to a specified accuracy.

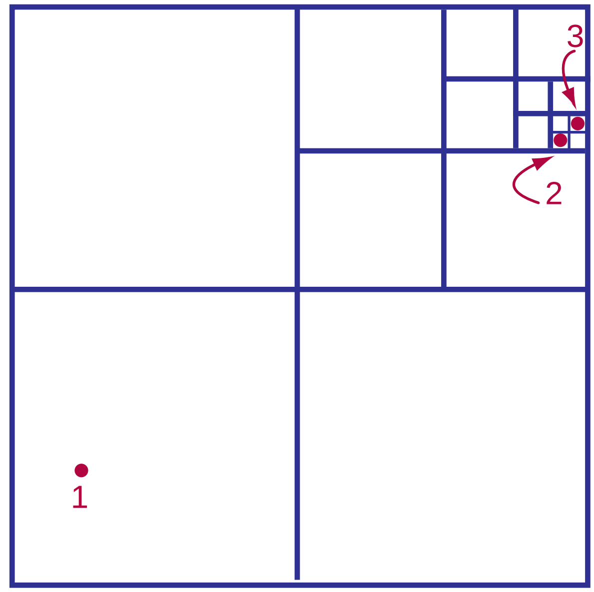

Figure 1: Barnes-Hut physical subdivision of space for a collection of three particles in two dimensions.

The same data structure is used in both methods, constructed as follows: Begin with a cell (square in two dimensions and cube in three dimensions) big enough to contain all the particles. Subdivide the cell into 2^d cells having half the side length of the original cell (where d is the number of dimensions). Discard cells that do not contain any particles. Stop the subdivision process on cells having exactly one particle. Recursively subdivide the cells that contain more than one particle. This recursive subdivision of the space into cells is naturally represented by a tree, which we shall refer to as the Barnes-Hut (BH) tree.

Figure 1 shows the Barnes-Hut physical subdivision of the space for a collection of three particles positioned as shown. The corresponding BH tree is shown in Figure 2. For convenience and simplicity, a two-dimensional problem is discussed but the results carry over to three-dimensional problems as well. In two dimensions, each cell is subdivided into four cells and the resulting structure is a quad-tree. In the example shown, the first subdivision separates particle 1 from particles 2 and 3. The next three subdivisions performed to separate particles 2 and 3 are not successful as one of the child cells at every level of the subdivision contains both the particles and the other three contain none. The recursive subdivision is continued until the particles 2 and 3 are separated.

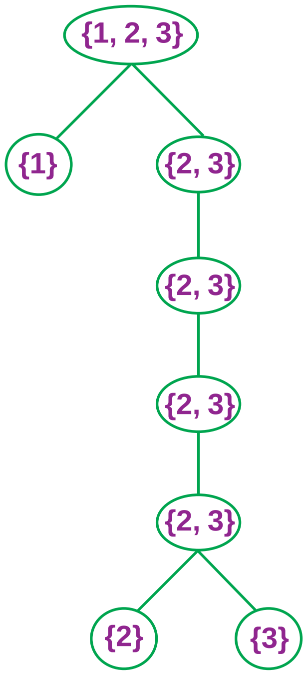

Figure 2: The Barnes-Hut tree corresponding to the physical subdivision of space in Figure 1.

From this example, it is clear that a large number of recursive subdivisions may be required to separate particles that are very close to each other. Let N be the number of particles in the system and let s be the smallest interparticle distance. We require $s > 0$ to avoid infinite interaction force. Let D be the length of a cell that can contain all the particles. Clearly, the worst-case path length of the BH tree is given by the worst-case path needed to separate the two particles which are closest to each other. The size of the smallest cell that can contain two particles s apart in two dimensions is $\frac{s}{\sqrt{2}}$ ($\frac{s}{\sqrt{3}}$ in three dimensions, see Figure 3). The paths separating the closest particles may contain recursive subdivisions until a cell of length smaller than $\frac{s}{\sqrt{2}}$ reached. Since each subdivision halves the length of the cells, the maximum path length is given by the smallest k for which

$$\frac{D}{2^k} < \frac{s}{\sqrt{2}}$$

$$k = \left\lceil \log \frac{\sqrt{2}D}{s} \right\rceil$$

In three dimensions,

$$k = \left\lceil \log \frac{\sqrt{3}D}{s} \right\rceil$$

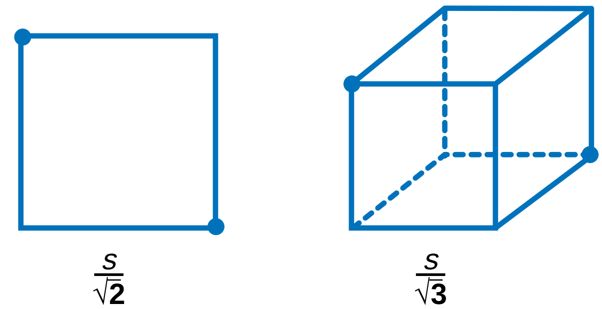

Figure 3: Smallest cells that could possibly contain two particles that are *s* apart in two and three dimensions.

In either case, the worst-case path length is $O(\log \frac{D}{s})$ and the number of nodes in the tree is bounded by $O(N \log \frac{D}{s})$.

Greengard assumes the length D of the cell containing all the particles is one, which can be achieved by appropriate scaling. Greengard's arguments can be summarized as follows: For a fixed machine precision, only certain classes of particle distributions can be modeled, independent of the algorithm used. Therefore, by restricting attention to only those particle distributions that can be modeled on a given machine, s has to be no less than the smallest floating point number representable. Thus, $\log \frac{D}{s}$ is bounded by a constant, termed p. The size of the tree is bounded by $O(pN)$. Greengard determines the running time of his algorithm in two dimensions to be $N(\alpha p^2 + \beta p + \gamma)$, where α, β and γ are constants.

The above arguments imply that the height of the tree is bounded by $O(p)$, a constant. Yet, we know that the height of a tree with N leaves and at most a constant number of children per node is $\Omega(\log N)$. How can this disparity be explained?

The problem lies in the assumption that the parameters D and s are entirely dependent on the spatial distribution of the particles and not related to the number of particles N. To understand why this assumption is invalid, consider the behavior of $\frac{D}{s}$ as a function of N.

To minimize the ratio $\frac{D}{s}$ for a fixed N, all the particles should be at a distance of s from their nearest neighbors. To see why, suppose this is not true. We can reduce D by 'moving-in' particles that are farther than s

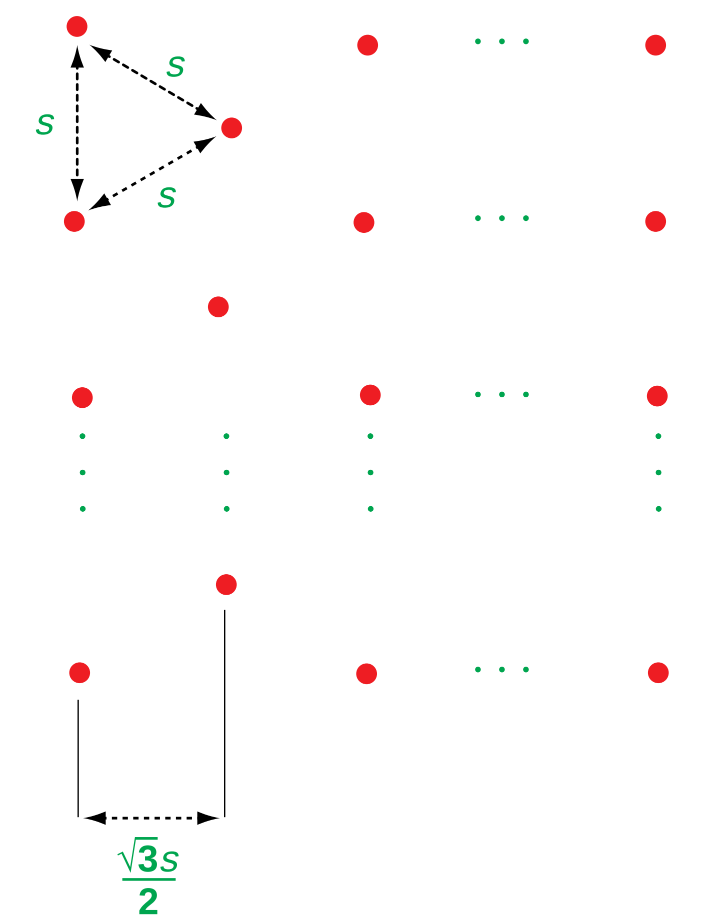

Figure 4: The configuration minimizing the ratio of the cell length containing all the particles and the smallest interparticle distance in two dimensions. The ratio is minimized when all particles are a distance *s* from their nearest neighbors.

from each other, while keeping s the same. Or, we can increase s by increasing the distance between particles that are s apart, keeping D unchanged. In either case, $\frac{D}{s}$ decreases, contradicting minimality. Furthermore, the particles must be packed as closely as possible. Figure 4 shows the configuration minimizing the ratio $\frac{D}{s}$ for a fixed N in two dimensions. Each particle has six nearest neighbors, all at a distance s. The particle is at the center of the hexagon formed by its nearest neighbors. The particles do not fit in a cell of size smaller than D x D. Adding the particles column-wise,

$$N = \left\lfloor \frac{D}{s} + 1 \right\rfloor + \left\lfloor \frac{D}{s} \right\rfloor + \left\lfloor \frac{D}{s} + 1 \right\rfloor + \ldots \quad \left(\left\lfloor \frac{2D}{\sqrt{3}s} + 1 \right\rfloor terms \right)$$

$$N \leq \frac{D}{s} \left[\frac{2D}{\sqrt{3}s} + 1 \right] + \frac{D}{\sqrt{3}s} + 1$$

$$N \leq \frac{2}{\sqrt{3}} \frac{D^2}{s^2} + \left(1 + \frac{1}{\sqrt{3}}\right) \frac{D}{s} + 1$$

$$\frac{D}{s} \geq c_1 \sqrt{N}$$

for some constant c_1. Since this is computed using the configuration minimizing the ratio $\frac{D}{s}$, the worst- case path length ($\log \frac{D}{s}$) is $\Omega(\log N)$. In three dimensions,

$$\frac{D}{s} \geq c_2 N^{\frac{1}{3}}$$

In either case,

$$\log \frac{D}{s} = \Omega(\log N)$$

Since $\log \frac{D}{s}$ is bounded by p, p is also $\Omega(\log N)$. The error in Greengard's proof was the assumption that p was independent of N.

How does this affect particle distributions that can be modeled on a machine with precision parameter p? It is already noted that not all distributions can be modeled for any given $N \geq 3$ because of precision limits. However, unless $p \geq c \log N$ (c a constant), no distribution can be modeled for that N. The very fact that we are able to run an N-body problem for a collection of N particles with precision parameter p implies that $p \geq c \log N$. Thus, p cannot be taken as a constant in the analysis of the running time of the algorithm and Greengard's algorithm is not $O(N)$. Greengard's time complexity in two dimensions is $N(\alpha p^2 + \beta p + \gamma)$, which is $\Omega(N \log^2 N)$. The three-dimensional complexity is $N(\alpha p^4 + \beta p^2 + \gamma)$, which is $\Omega(N \log^4 N)$.

Next, let us investigate how large $\frac{D}{s}$ can be for a fixed N. For any $N \geq 3$ particles, $\frac{D}{s}$ can be made arbitrarily large by reducing the distance between the closest particles (thus reducing s), or by increasing the spread of the particles (thus increasing D). Hence, the worst-case path length does not have an upper bound as a function of the number of particles and is entirely dependent upon the spatial distribution of the particles. This immediately implies that the size of the BH tree is unbounded and can be arbitrarily large for a fixed N. Since both the Greengard's and Barnes-Hut algorithms construct and visit each node in the BH tree at least once, these algorithms are unbounded for arbitrary distributions.

Clearly, not all particle distributions can be modeled on a given machine due to precision limits. But, an algorithm whose running time depends upon the distribution is undesirable. An analogy can be drawn to a sorting algorithm whose running time depends on the size of the numbers to be sorted. The complexity of a sorting algorithm is $O(n \log n)$, provided basic operations on the numbers to be sorted (like comparison, copying) can be accomplished in constant time. The complexity of the algorithm does not remain $O(n \log n)$ if this assumption is not valid. But, a sorting algorithm with running time as a function of the size of the numbers to be sorted is undesirable. Similarly, it is reasonable to assume that the distribution of the particles is representable in a given machine but algorithms whose running times depend on the distribution are undesirable.

3 A Modified Data Structure

The BH tree can contain a path on which every node represents the same set of particles, though each node represents a cell of a different size. Such a path can be arbitrarily large irrespective of the total number of particles. Each node on the path represents a cell of exactly half the length of the cell represented by its parent. Our intent is to rectify this unbounded nature of the BH tree.

Let $v_1, v_2, \ldots, v_k$ ($k \geq 2$) be a maximal path in the BH tree such that each node of the path represents the same set of particles. The maximality of the path ensures that v_1's parent has more particles than v_1 and no child of v_k has the same particles as v_k. Since only cells having more than one particle are subdivided, it is imperative that v_k have at least two child nodes. We can also assume without loss of generality that v_1 has a parent. Otherwise, v_1 has to be the root of the tree, thus containing all the particles in the system. By the property of the path v_1, $v_2, \ldots, v_k$, v_k also contains all the particles in the system. This simply means that our choice of the initial cell is too large for the system of particles and a cell $\frac{1}{2^k}^{th}$ length of it (this is the cell represented by v_k) can contain the entire system. In this case, we can safely make the subtree rooted at v_k be the BH tree. Therefore, it can be assumed that v_1 always has a parent. Furthermore, v_i is the only child of v_i–1 ($1 < i \leq k$).

Consider such a maximal path $v_1, v_2, \ldots, v_k$. Every node on such a path represents the same particles, but using cells of different sizes. Nodes in the BH tree are used to store aggregate information on the collection of particles they represent. For example, the Barnes-Hut method keeps track of the total mass and the center of mass of the collection of particles. Greengard's method computes the multipole expansion of the collection of particles. Since every node on such a path represents the same particles, they all contain the same information. Therefore, a modified tree obtained by eliminating this redundancy should contain the same information as the BH tree.

Consider the tree obtained by replacing every such maximal path by the last node on the path. The modified tree has N leaves, one per particle and the root of the tree contains all N particles, just as in the Barnes-Hut tree. The only deviation is that no internal node can have the same particles as one of its children. Since the number of particles contained by a node is the sum of the number of particles contained by its children, this translates to the condition that each internal node has at least two children. Let $S(N)$ be the number of nodes in the modified tree for a collection of N particles in d dimensions.

$$S(N) = 1 + \sum_{i=1}^{k} S(N_i) \qquad (2 \leq k \leq 2^d)$$

$$\sum_{i=1}^{k} N_i = N$$

$$S(1) = 1$$

These equations are satisfied by $S(N) \leq 2N - 1$, and the size of the tree is bounded by $O(N)$ in any dimensions. A tree containing N leaves has a size $\Omega(N)$, proving the optimality of the modified tree. Since each child contains at least one particle less than its parent, the path length is also bounded by $O(N)$.

3.1 The Barnes-Hut method using the modified data structure

In the Barnes-Hut method [4], the BH tree is traversed once for every particle in the system to approximate the force acting on the particle due to the rest of the system. The force on any particle p is approximated using the following recursive calculation: Let l be the length of the cell currently being processed. Let d be the distance between the particle and the center of mass of the cell under consideration. If $\frac{l}{d} < \theta$, where $0 \leq \theta < 1$

is a prespecified accuracy criterion, the cell is treated as a single particle of equivalent mass located at the center of mass for the purpose of force calculation. Otherwise, the children of the cell are examined recursively to com- pute the force on p. The force calculation starts by examining the root cell. This calculation is repeated once for every particle in the system.

Let $v_1, v_2, \ldots, v_k$ be a maximal path in the BH tree such that each node contains the same particles and let v_0 be the parent of v_1. In the modified tree, v_k is the child of v_0. Suppose that the node v_1 is reached while traversing the Barnes-Hut tree to compute the force on a particle p. Either the tree traversal stops at some v_i ($1 \leq i \leq k$) or the traversal proceeds to the children of v_k. Let $l(v_i)$ be the length of the cell, $cm(v_i)$ be the center of mass, and $M(v_i)$ be the total mass of the particles in the cell represented by node v_i. Note that

$$M(v_1) = M(v_2) = \ldots = M(v_k)$$
$$cm(v_1) = cm(v_2) = \ldots = cm(v_k)$$
$$l(v_1) = 2l(v_2) = 2^2\, l(v_3) = \ldots = 2^{k-1}\, l(v_k)$$

If the traversal stopped at some v_i ($1 \leq i \leq k$),

$$\frac{l(v_i)}{d(p, cm(v_i))} < \theta,$$

where $d(p, cm(v_i))$ is the distance from p to the center of mass of the cell represented by v_i and θ is the accuracy criterion. Since v_j is the only child of v_{j-1} ($1 < j \leq k$), the force contributed by the subtree rooted at v_1 is the force between p and a mass of $M(v_i)$ located at $cm(v_i)$. In traversing the modified tree, v_k is reached instead of v_1. Since $k \geq i$,

$$\frac{l(v_k)}{d(p, cm(v_k))} = \frac{1}{2^{k-i}} \frac{l(v_i)}{d(p, cm(v_i))} < \theta$$

The force contributed by the subtree rooted at v_k is the force between p and a mass of $M(k)$ located at $cm(v_k)$, which is the same as the force contributed by the subtree under v_1 in the Barnes-Hut tree.

If the Barnes-Hut tree traversal proceeds to the children of v_k, the same happens in the modified tree also. The force contributed by the subtree rooted at v_1 in the Barnes-Hut tree is the force contributed by the subtree rooted at v_k, which is the same for both trees. In either case, the force computations give the same result on both trees. The modified tree rectifies the unbounded nature of the Barnes-Hut tree without changing the force calculations of the Barnes-Hut algorithm. However, it improves the Barnes-Hut algorithm in two important ways: First, the running time of the algorithm is reduced. In fact, the worst-case traversal on the Barnes-Hut tree is unbounded as the BH tree is unbounded. The modified tree also allows for more accurate error estimation. The error in approximating the force between a particle p and a cluster of particles by treating the cluster as a single particle of equivalent mass located at the center of mass is proportional to $\left(\frac{dr}{r}\right)^2$, where dr is the radius of the cluster and r is the distance of its center of mass from p. In the Barnes-Hut algorithm, the error created by treating the cell represented by node v_i as a single particle is therefore proportional to

$$\left(\frac{l(v_i)}{d(p, cm(v_i))}\right)^2$$

If $v_1, v_2, \ldots, v_k$ is a maximal path in the Barnes-Hut tree with every node containing the same particles and the Barnes-Hut tree traversal stopped at some v_i ($1 \leq i \leq k$), the error made is computed to be proportional to

$$\left(\frac{l(v_i)}{d(p, cm(v_i))}\right)^2$$

This is an overestimation of the error because the radius of the cluster of particles is taken to be $l(v_i)$ whereas the radius is in fact bounded by $l(v_k) = \frac{l(v_i)}{2^{k-i}}$ A traversal on the modified tree computes the same force with an error estimate proportional to

$$\left(\frac{l(v_k)}{d(p, cm)v_k))}\right)^2 = \frac{1}{2^{2(k-i)}}\left(\frac{l(v_i)}{d(p, cm(v_i))}\right)^2$$

The error estimate at this node is thus improved by a factor of $2^{2(k-i)}$.

3.2 Greengard's method using the modified data structure

Greengard's fast multipole algorithm [7] is a two-pass procedure on the BH tree. The first pass is a bottom-up traversal of the tree in which a p-term multipole expansion is formed at every node of the tree, where p is a precision parameter. The multipole expansions at the leaves are computed directly. At any internal node, the multipole expansion is formed by shifting the multipole expansions of the child nodes to the center of the cell represented by the node and adding them together. In the second pass, the tree is traversed top-down to compute the local expansions at every node. The local expansion at a node is formed by shifting the local expansion at the parent node to its center, shifting the multipole expansions of the *well-separated* children of the nearest neighbors of the parent of the node to its center and adding them together. Finally, the local expansions at every leaf are evaluated to compute the approximate cumulative force on each particle. For a detailed description of Greengard's algorithm, see [7].

Consider a run of the Greengard's algorithm on the BH tree containing a path $v_1, v_2, \ldots, v_k$, where each node represents the same particles. Since v_i is the only child of v_{i-1} $(1 < i \leq k)$, the multipole expansion at v_{i-1} is formed by shifting the multipole expansion of v_i to the center of the cell represented by v_{i-1}. The multipole expansions at these nodes are merely translations of one another. Since $v_1, v_2, \ldots, v_k$ is a chain, the multipole expansions at these nodes are useful only to compute the multipole expansion of v_1's parent. However, the contribution by v_1's multipole expansion to the multipole expansion of its parent can be directly obtained by shifting the multipole expansion of v_k to the center of the cell represented by the parent of v_1. Thus, computing the multipole expansions at $v_1, v_2, \ldots, v_{k-1}$ is unnecessary and is avoided by the modified tree. A similar argument shows that the correct local expansions at the leaves can be obtained using the modified tree.

In the multipole algorithm designed to run on the modified tree, the precision parameter p is a constant since it can be chosen independent of N. In the Greengard's algorithm, p has a lower bound of log N. This is because p is also used as an upper bound on the worst-case path length ($\log \frac{D}{s}$) of the BH tree, which has a lower bound of log N. Therefore, p cannot be chosen independent of N and is also a function of the distribution of the particles. In the multipole algorithm on the modified tree, the precision parameter is merely a function of the desired accuracy of the force calculations chosen independent of the number and distribution of the particles.

The new algorithm consists of two traversals of the modified tree. Computing the p-term multipole/local expansions at a node take constant time. Evaluating a p-term local expansion for every particles also takes constant time. Since the number of nodes in the modified tree is $O(N)$, running the multipole algorithm on the modified tree takes $O(N)$ time. This is irrespective of the distribution of the particles.

The running time of this algorithm depends on the complexity of the tree creation and the complexity of performing the force calculations. It is already noted that the force computations can be performed in $O(N)$ time on the modified tree. In the next section, we show that the modified tree can be created in $O(N \log N)$ time in two dimensions and in $O(N \log^2 N)$ time in three dimensions. Thus, the new multipole algorithm has a complexity of $O(N \log N)$ in two dimensions and $O(N \log^2 N)$ in three dimensions.

4 Creating the Modified Tree

This section describes an algorithm to construct the modified tree for a collection of N particles. The concepts are illustrated with two-dimensional figures for convenience, but the results are applicable to three-dimensional problems as well. First, some terminology:

The physical space containing the particles is subdivided using cells. A cell is completely determined by the length of an edge of the cell and the position of one of the corners of the cell. The corner is chosen to be the point in the cell with the smallest value for each coordinate. In two dimensions, this is the lower, leftmost corner. Let l be the length of a cell. In order to describe the subcells of this cell, the corner of this cell is taken to be the origin. The cell contains 2^{kd} cells of length $\frac{l}{2^k}$. The cells are positioned at $(i \frac{l}{2^k}, j \frac{l}{2^k})$ $(0 \leq i, j < 2^k - 1)$ in two dimensions. A line is called a *k-boundary* if it contains an edge of a cell of length $\frac{l}{2^k}$. There are $2^k + 1$ lines parallel to each axis and spaced $\frac{l}{2^k}$ apart that are *k-boundaries*. The intersections of the *k-boundaries* determine the cells of size $\frac{l}{2^k}$. Any *k-boundary* is also a *j-boundary* for every $j > k$. See Figure 5. In three dimensions, a *k-boundary* is a plane containing a surface of a cell of size $\frac{l}{2^k}$. Note that the description of the subcells and the boundaries is relative to a cell.

A simple recursive algorithm for creating the modified tree for a cell containing a collection of particles is given below:

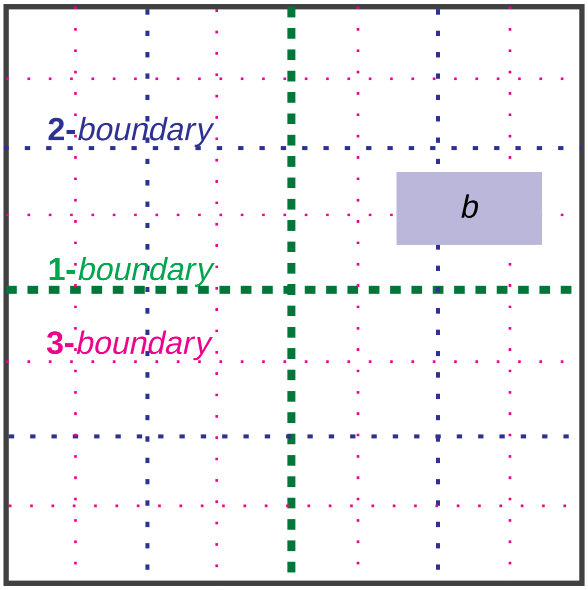

Figure 5: A cell of length *l* and the smallest box *b* enclosing all the particles in this cell. The big dashed lines are 1-*boundaries,* the small dashed lines are 2-*boundaries* and the dotted lines are 3-*boundaries*. 2-*boundaries* are also 3-*boundaries* and 1-*boundaries* are also 2-*boundaries* and 3-*boundaries*.

BuildTree(c)

1. Find the smallest cell c' contained in c that still contains all the particles contained in cell c.
2. If c contains no particles, return '*empty tree*'.
3. If c contains exactly one particle, return the one node tree c.
4. Split the cell c' into 2^d subcells.
5. For each subcell sc of c', *BuildTree*(sc).
6. Return the tree obtained by joining all the trees obtained in the previous step, with c' as the root of the tree.

BuildTree is initially called with a cell large enough to contain all the particles in the system. The running time of *BuildTree* can be computed by the amount of work done at every node of the modified tree, which is steps 1-4 and 6. Steps 4 and 6 require a constantamount of work at every node of the modified tree. Steps 2 and 3 can be accomplished as a byproduct of Step 1, as we shall see later. Step 1 can be accomplished as follows:

Let l be length of the cell c passed as input to *BuildTree*. Any cell smaller than c but contained in c has length $\frac{l}{2^k}$ for some $k > 0$. By a suitable transformation, the corner of c is chosen to be the origin. Let b be the smallest box (a rectangle in two dimensions) containing all the particles of c. The rectangle is given by (x_{min}, y_{min}), (x_{max}, y_{min}), (x_{max}, y_{max}), (x_{min}, y_{max}), where x_{min} is the smallest x coordinate of all the particles in c etc. The smallest cell in c containing all the particles should also contain the box b. A cell of size $\frac{l}{2^k}$ encloses b iff no *k-boundary* passes through b (see Figure 5). The smallest cell enclosing b is of size $\frac{l}{2^{k-1}}$,where k is the smallest integer for which a *k-boundary* passes through b. To determine this, we can examine boundaries parallel to each coordinate axis in turn.

Consider boundaries parallel to the y-axis. These can be specified by their distance from the y-axis. The family of *k-boundaries* is specified by $i \frac{l}{2^k}$,$0 \le i \le 2^k$.

We need to find the smallest integer k such that a *k- boundary* parallel to y-axis passes through b, i.e. the smallest k such that $x_{min} < i \frac{l}{2^k} < x_{max}$ for some i. By minimality of k, only one *k-boundary* passes through b. Let j be the smallest integer such that $\frac{l}{2^j} < (x_{max} - x_{min})$. $j = \left\lceil \log_2 \frac{l}{x_{max} - x_{min}} \right\rceil$. There is at least 1 and at most 2 *j-boundaries* passing through b. These boundaries are given by $h_1 = \left\lceil \frac{2^j x_{min}}{l} \right\rceil \frac{l}{2^j}$ and $h^2 = \left\lfloor \frac{2^j x_{max}}{l} \right\rfloor \frac{l}{2^j}$. Since $k \le j$, any *k-boundary* is also a *j-boundary*, forcing the *k-boundary* passing through b to coincide with h_1 or h_2. Let a be $\left\lceil \frac{2^j x_{min}}{l} \right\rceil$. $h_1 = a \frac{l}{2^j}$ and $h_2 = h_1$ or $(a + 1) \frac{l}{2^j}$. If $h_2 \ne h_1$, let a' be the even integer among a and a+1. Otherwise, let a' be equal to a. It is clear that $j - k$ is equal to the highest power of 2 that divides a'. One way to find this is $j - k = \log_2(1 + \{a' \oplus (a' - 1)\}) - 1$. Since all the above operations take constant time, the smallest cell contained in c enclosing the box b can be determined in constant time.

It is already established that the modified tree has $O(N)$ nodes. The tree is created top-down starting at the root. At each node, the particles with the smallest and the largest coordinates in each dimension (x_{min}, x_{max}, y_{min} and y_{max} in two dimensions) are computed to identify the smallest box enclosing all the particles represented by the node. The smallest cell enclosing this box is computed and the children of the node determined in constant time. Note that the particles are not distributed among the child nodes. Such a distribution would result in $O(N^2)$ time for tree creation. Distributing the particles to the child nodes is not necessary provided we can determine the particles with extreme coordinates in the child nodes. Except for this task, the rest of the computations are done in constant time per node, for a total of $O(N)$ time.

In *BuildTree*, we also need to determine cases where the cell contains exactly one particle or none. This can be determined as a byproduct of the computation of the smallest box b containing all the particles in the cell. If $x_{min} = x_{max}$ and $y_{min} = y_{max}$, the cell contains exactly one particle. If the answer to any of the 4 queries is < none >, the cell is empty and can be discarded.

Finding the points with extreme coordinates can be translated to a range query problem, stated as follows: Given N points, set up a data structure to answer queries of the form 'which point has the smallest x-coordinate among the points that lie in a given square?' efficiently. Since the modified tree has $O(N)$ nodes and we require four such queries per node (eight in three dimensions), the number of queries is $O(N)$. The range query problem can be solved in $O(N \log N)$ preprocessing time and $O(\log N)$ query time in two dimensions. The corresponding times for three dimensions are $O(N \log^2 N)$ and $O(\log^2 N)$ respectively. The solution makes use of *Priority Search Trees* [12] and is omitted in the interest of brevity. The time required for preprocessing and answering $O(N)$ queries is $O(N \log N)$ in two dimensions and $O(N \log^2 N)$ in three dimensions. Consequently, the modified tree can be created in $O(N \log N)$ time in two dimensions and $O(N \log^2 N)$ time in three dimensions.

5 Conclusions

The N-body problem has a lower bound of $\Omega(N)$ to compute all pairwise interactions. In light of the proof that the Greengard's method is not $O(N)$, the fastest distribution-independent algorithm has a complexity of $O(N \log N)$ in two dimensions and $O(N \log^2 N)$ in three dimensions. This complexity is mainly due to the hierarchical tree creation. A linear time algorithm to update the tree structure results in an algorithm with complexity matching the lower bound. The pos- sibility of such an algorithm remains to be investigated.

Acknowledgments

The authors wish to thank Dr. Ravi Janardhan for suggesting the use of priority search trees to solve range query problems.

References

[1] S. Aluru, *Distribution-independent hierarchical N-body methods*, Ph.D. thesis, Iowa State University, 1994.

[2] A.W. Appel, An efficient program for many-body simulation, *SIAM J. Sci. Stat. Comput.*, 6 (1985) 85-103.

[3] J. Barnes, A modified tree code: Don't laugh; It runs, *J. Comput. Phys.*, 87 (1990) 161-170.

[4] J. Barnes and P. Hut, A hierarchical $O(N \log N)$ force-calculation algorithm, *Nature, 324* (1986) 446-449.

[5] S. Bhatt, M. Chen, C.Y. Len and P. Liu, Abstractions for parallel N-body simulations, Tech. Rep. DCS/TR-895, Yale University, 1992.

[6] K. Esselink, The order of Appel's algorithm, *Info. Proc. Letters*, 41 (1992) 141-147.

[7] L. Greengard, The rapid evaluation of potential fields in particle systems, MIT Press, Cambridge, MA, 1988.

[8] L. Hernquist, Vectorization of tree traversals, *J. Comp. Phys., 87* (1990) 137-147.

[9] R.W. Hockney and J.W. Eastwood, *Computer simulation using particles*, McGraw-Hill, New York, 1981.

[10] J. Katzenelson, Computational structure of the N-body problem, *SIAM. J. Sci. Stat. Comput.*, 10 (1989) 787-915.

[11] J. Makino, Vectorization of a treecode, *J. Comp. Phys., 87* (1990) 148-160.

[12] E.M. Mc Creight, Priority Search Trees, *SIAM J. Comput.* (1985) 257-268.

[13] L. Greengard and V. Rokhlin, A fast algorithm for particle simulations, *J. Comp. Phys.*, 73 (1987) 325-348.

[14] J.K. Salmon, *Parallel hierarchical N-body methods*, Ph.D. thesis, California Institute of Technology, 1990.

[15] J.P. Singh, *Parallel hierarchical N-body methods and their implications for multiprocessors*, Ph.D. thesis, Stanford University, 1993.

[16] J.P. Singh, C. Holt, T. Totsuka, A. Gupta and J.L. Hennesy, Load balancing and data locality

in hierarchical *N*-body methods, *Journal of Parallel and Distributed Computing*, to appear.

[17] M.S. Warren and J.K. Salmon, Astrophysical *N*-body simulations using hierarchical tree data structures, *Proc. Supercomputing* ’92 (1992) 570- 576.

[18] M.S. Warren and J.K. Salmon, A parallel hashed oct-tree *N*-body algorithm, *Proc. Supercomputing ’93* (1993) 1-12.

[19] F. Zhao and L. Johnsson, The parallel multipole method on the connection machine, *SIAM. J. Sci. Stat. Comput.*, 12 (1991) 1420-1437.

* * * * * * * * * * * *